Deep Deterministic Policy Gradient

Deep Reinforcement Learning

Sai Sampath Kedari
sampath@umich.edu

Contents

1 Deep Deterministic Policy Gradient

Deep Q-Networks extend Q-learning to large state spaces using nonlinear function approximation, experience replay, and target networks. However, DQN is restricted to finite, relatively low-dimensional action spaces because its Bellman optimality target requires

$$\max_{a' \in \mathcal{A}} \hat{q}(S_{t+1}, a', \bar{w}).$$

For a continuous action space

$$\mathcal{A} \subseteq \mathbb{R}^m,$$

this maximization becomes a continuous optimization problem that would have to be solved repeatedly during action selection and Bellman backups. Discretizing the action space is generally impractical in high dimensions because the number of discrete actions grows exponentially with the action dimension.

Deep Deterministic Policy Gradient (DDPG) addresses this problem by combining the deterministic policy gradient with the stabilization mechanisms introduced by DQN. DDPG is a model-free, off-policy actor–critic algorithm for continuous action spaces.

The actor is a deterministic policy

$$\mu_\theta : \mathcal{S} \to \mathcal{A}, \qquad A = \mu_\theta(S),$$

which directly produces a continuous action. The critic is a parameterized action-value function

$$\hat{q}(s, a, w) \approx Q^{\mu_\theta}(s, a)$$

that evaluates actions under the current deterministic policy.

The actor therefore replaces the explicit continuous-action maximization that would otherwise be required by Q-learning:

$$\arg\max_{a \in \mathcal{A}} \hat{q}(s, a, w) \quad \longrightarrow \quad \mu_\theta(s).$$

The actor is improved using the deterministic policy gradient, while the critic is learned off-policy from transitions stored in an experience replay buffer. As in DQN, target networks are used to stabilize the bootstrapped critic targets; because the critic target depends on the next action as well as its value, DDPG maintains both a target actor and a target critic.

Thus, DDPG can be viewed as combining

$$\boxed{\text{Deterministic Policy Gradient} + \text{DQN stabilization machinery}}$$

to obtain a deep, off-policy actor–critic method for continuous control.

1.1 Critic Update

For the deterministic policy

$$A = \mu_\theta(S), \qquad \mu_\theta : \mathcal{S} \to \mathcal{A},$$

the critic approximates its state-action value function,

$$\hat{q}(s, a, w) \approx Q^{\mu_\theta}(s, a).$$

For a stochastic policy π, the action-value function satisfies the Bellman expectation equation

$$Q^{\pi}(s,a) = \mathbb{E}_{(S_{t+1},R_{t+1})\sim p(\cdot,\cdot|s,a)}\left[R_{t+1} + \gamma \mathbb{E}_{A_{t+1}\sim\pi(\cdot|S_{t+1})}\left[Q^{\pi}(S_{t+1},A_{t+1})\right]\right]. \tag{1}$$

For the deterministic policy μ_θ,

$$A_{t+1} = \mu_\theta(S_{t+1}),$$

so the inner expectation disappears:

$$\boxed{Q^{\mu_\theta}(s,a) = \mathbb{E}_{(S_{t+1},R_{t+1})\sim p(\cdot,\cdot|s,a)}\left[R_{t+1} + \gamma Q^{\mu_\theta}\left(S_{t+1},\mu_\theta(S_{t+1})\right)\right].} \tag{2}$$

The expectation is taken with respect to the environment dynamics alone. This means that it is possible to learn Q^{μ_θ} off-policy, using transitions which are generated from a different stochastic behaviour policy β.

For a sampled transition

$$(S_i, A_i, R_{i+1}, S_{i+1}),$$

the corresponding one-step bootstrapped target is

$$\boxed{Y_i = R_{i+1} + \gamma \hat{q}\left(S_{i+1},\mu_\theta(S_{i+1}),w\right).} \tag{3}$$

For a replay mini-batch

$$\mathcal{B} = \{(S_i, A_i, R_{i+1}, S_{i+1})\}_{i=1}^{B} \sim \text{Uniform}(\mathcal{D}),$$

the critic is trained by minimizing

$$\boxed{\mathcal{L}_{\text{critic}}(w) = \frac{1}{2B}\sum_{i=1}^{B}\left[Y_i - \hat{q}(S_i,A_i,w)\right]^2.} \tag{4}$$

The critic parameters are updated by

$$w \leftarrow w + \frac{\alpha_q}{B}\sum_{i=1}^{B}\left[Y_i - \hat{q}(S_i,A_i,w)\right]\nabla_w \hat{q}(S_i,A_i,w). \tag{5}$$

Here Y_i is treated as constant during the update.

1.2 Actor Update

From the deterministic policy gradient report, the local actor surrogate for the current policy μ_{old} is

$$\boxed{J_{\text{actor}}(\theta;\theta_{\text{old}}) = \frac{1}{1-\gamma}\mathbb{E}_{S\sim d^{\mu_{\text{old}},\rho}}\left[Q^{\mu_{\text{old}}}\left(S,\mu_\theta(S)\right)\right].} \tag{6}$$

At $\theta = \theta_{\text{old}}$, the gradient of this surrogate is exactly the deterministic policy gradient.

DDPG makes two practical replacements. The on-policy discounted state-visitation distribution is replaced by the state distribution induced by the behaviour policy,

$$d^{\mu_{\text{old}},\rho} \longrightarrow \rho^{\beta},$$

where ρ^β denotes the replay-state distribution induced by the behaviour policies used to collect the stored experience. The true action-value function is replaced by the learned critic,

$$Q^{\mu_{\text{old}}}(s,a) \longrightarrow \hat{q}(s,a,w).$$

Dropping the positive constant $1/(1-\gamma)$, the DDPG actor objective is therefore

$$\boxed{J_{\text{actor}}(\theta;w) = \mathbb{E}_{S\sim\rho^\beta}\left[\hat{q}\left(S,\mu_\theta(S),w\right)\right].} \tag{7}$$

This is an off-policy approximation to the actor surrogate derived above: the state distribution is changed from $d^{\mu_{\text{old}},\rho}$ to ρ^β, and $Q^{\mu_{\text{old}}}$ is approximated by the critic.

During the actor update, w is held fixed, giving

$$\boxed{\nabla_\theta J_{\text{actor}}(\theta;w) = \mathbb{E}_{S\sim\rho^\beta}\left[\nabla_\theta\mu_\theta(S)\ \nabla_a\hat{q}(S,a,w)|_{a=\mu_\theta(S)}\right].} \tag{8}$$

Using the same replay mini-batch $\mathcal{B}$, the objective is estimated by

$$\widehat{J}_{\text{actor}}(\theta;w) = \frac{1}{B}\sum_{i=1}^{B}\hat{q}\left(S_i,\mu_\theta(S_i),w\right), \tag{9}$$

with gradient

$$\nabla_\theta\widehat{J}_{\text{actor}}(\theta;w) = \frac{1}{B}\sum_{i=1}^{B}\nabla_\theta\mu_\theta(S_i)\ \nabla_a\hat{q}(S_i,a,w)|_{a=\mu_\theta(S_i)}. \tag{10}$$

The actor parameters are updated by gradient ascent,

$$\theta \leftarrow \theta + \alpha_\mu\nabla_\theta\widehat{J}_{\text{actor}}(\theta;w). \tag{11}$$

In implementation, automatic differentiation computes this gradient directly through

$$\theta \longrightarrow \mu_\theta(S_i) \longrightarrow \hat{q}\left(S_i,\mu_\theta(S_i),w\right),$$

with the critic parameters held fixed.

1.3 Target Networks

The critic target above,

$$Y_i = R_{i+1} + \gamma\hat{q}\left(S_{i+1},\mu_\theta(S_{i+1}),w\right),$$

depends on both learned networks. Updating w changes the value assigned to the next state-action pair, while updating θ changes the next action at which that value is evaluated. The regression targets can therefore change rapidly as learning proceeds.

As in DQN, DDPG stabilizes the bootstrapped targets using slowly moving target networks. Because the target depends on both the next action and its value, DDPG maintains a target actor and a target critic,

$$\mu_{\bar{\theta}}(s), \qquad \hat{q}(s,a,\bar{w}).$$

The critic target is then

$$\boxed{Y_i = R_{i+1} + \gamma \hat{q}\left(S_{i+1}, \mu_{\bar{\theta}}(S_{i+1}), \bar{w}\right).} \tag{12}$$

The online actor and critic may change at every learning step, while the functions used to construct Y_i change much more slowly. The target parameters track their online counterparts through Polyak averaging:

$$\boxed{\bar{w} \leftarrow \tau w + (1-\tau)\bar{w}, \qquad \bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}, \qquad 0 < \tau \ll 1.} \tag{13}$$

1.4 Exploration Policy

The deterministic actor

$$A_t = \mu_\theta(S_t)$$

does not provide exploration by itself. Since DDPG is off-policy, the behaviour policy used to collect experience may differ from the deterministic policy being learned.

DDPG therefore adds exploration noise to the actor output:

$$\boxed{A_t = \mu_\theta(S_t) + \mathcal{N}_t.} \tag{14}$$

The resulting transition

$$(S_t, A_t, R_{t+1}, S_{t+1})$$

is stored in the replay buffer. Thus the behaviour policy generates the data, while the actor update itself uses the deterministic action

$$\mu_\theta(S_i).$$

In the original DDPG experiments, temporally correlated Ornstein–Uhlenbeck noise was used for physical control tasks, although the algorithm is not tied to this particular noise process.

1.5 DDPG Algorithm

The critic update, actor update, target networks, and exploration policy now combine into the complete DDPG learning procedure.

At each environment step, the deterministic actor is perturbed by exploration noise,

$$A_t = \mu_\theta(S_t) + \mathcal{N}_t,$$

and the resulting transition

$$(S_t, A_t, R_{t+1}, S_{t+1})$$

is appended to the replay buffer $\mathcal{D}$.

A mini-batch of transitions is then sampled uniformly from $\mathcal{D}$. The target actor and target critic construct the bootstrapped critic targets; the online critic is fitted to those targets; the online actor is improved through the online critic using the deterministic policy gradient; and both target networks are finally moved toward their online counterparts by Polyak averaging.

Thus each learning step follows

$$\boxed{\text{collect} \longrightarrow \text{replay} \longrightarrow \text{critic update} \longrightarrow \text{actor update} \longrightarrow \text{target update.}}$$

Figure 1 shows this procedure as the temporal training loop. Figure 2 shows the same algorithm from the computation perspective, separating the target-network path used for critic regression from the online actor–critic path used for policy improvement.

Initialize online actor θ and online critic w, target copies $\bar{\theta} \leftarrow \theta$, $\bar{w} \leftarrow w$, replay buffer $\mathcal{D} \leftarrow \varnothing$.

$M\times$ **for** episode $= 1, \ldots, M$

Initialize a noise process $\mathcal{N}$ for exploration; receive initial state S_1.

$T\times$ **for** $t = 1, \ldots, T$

1. Select $A_t = \mu_\theta(S_t) + \mathcal{N}_t$, execute it, observe (S_{t+1}, R_{t+1}), and append $(S_t, A_t, R_{t+1}, S_{t+1})$ to $\mathcal{D}$.

2. Sample a mini-batch

$$\mathcal{B}_t = \{(S_i, A_i, R_{i+1}, S_{i+1})\}_{i=1}^{B} \sim \text{Uniform}(\mathcal{D}).$$

3. Compute targets from the *target* actor and critic:

$$Y_i = R_{i+1} + \gamma\, \hat{q}\big(S_{i+1}, \mu_{\bar{\theta}}(S_{i+1}), \bar{w}\big).$$

4. Update the *online* critic:

$$w \leftarrow w + \frac{\alpha_q}{B} \sum_{i=1}^{B} \left[Y_i - \hat{q}(S_i, A_i, w)\right] \nabla_w \hat{q}(S_i, A_i, w).$$

5. Update the *online* actor, with w held fixed:

$$\theta \leftarrow \theta + \frac{\alpha_\mu}{B} \sum_{i=1}^{B} \nabla_\theta \mu_\theta(S_i)\, \nabla_a \hat{q}(S_i, a, w)\big|_{a=\mu_\theta(S_i)}.$$

6. Update the target networks:

$$\bar{w} \leftarrow \tau w + (1-\tau)\bar{w}, \qquad \bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}.$$

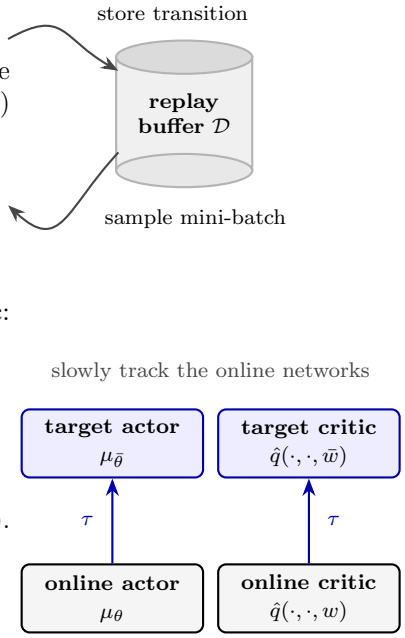

Figure 1: The DDPG training loop. At each environment step, the online actor selects a continuous action with added exploration noise and stores the resulting transition in the replay buffer. A mini-batch is then sampled to update the online critic from target-network Bellman targets and to update the online actor through the critic. Finally, the target actor and target critic are moved slowly toward their online counterparts by Polyak averaging. This cycle repeats throughout training.

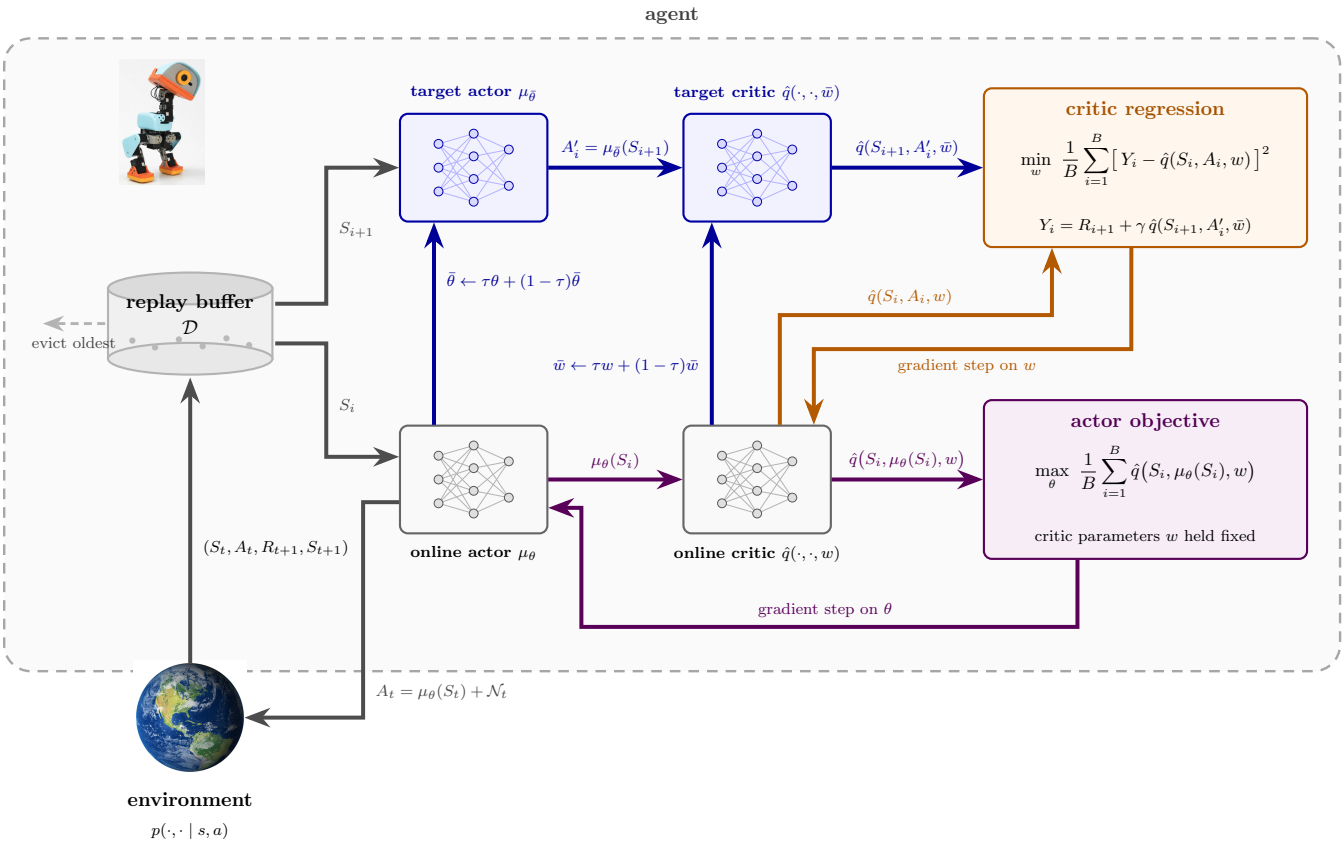

Figure 2: DDPG as two computation paths driven by replayed transitions. The target path (top) uses S_{i+1}, the target actor, and the target critic to construct the bootstrapped critic target Y_i. The online critic is then trained to match this target using the replayed pair (S_i, A_i). The actor path (bottom) uses S_i, the online actor, and the online critic to maximize $\hat{q}(S_i, \mu_\theta(S_i), w)$, with the critic parameters held fixed. The target actor and target critic slowly track their online counterparts by Polyak averaging, while the noisy online actor generates new experience for the replay buffer.